A Feature-based Sampling Method to Detect Anomalous Patterns in High Dimensional Datasets

Minh Quoc Nguyen
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
quocminh@cc.gatech.edu

Leo Mark
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
leomark@cc.gatech.edu

Edward Omiecinski
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
edwardo@cc.gatech.edu

ABSTRACT

We introduce a feature-based sampling method to detect anomalous patterns. By recognizing that an observation is considered normal because there are many observations similar to it, we formally define the problem of anomalous pattern detection. The properties of normal and anomalous patterns allow us to devise a generic framework using the sampling method to quickly prune the normal observations. Observations that can not form significant patterns are anomalous. Rules that are learned from the dataset are used to construct the patterns for which we compute a score function to measure the interestingness of the anomalous patterns. Experiments using the KDD Cup 99 dataset show that our approach can discover most of the attack patterns. Those attacks are in the top set of anomalous patterns and have a higher score than the patterns of normal connections. The experiments also show that the algorithm can run in near linear time.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Applications - Data Mining

Keywords

Data Mining, Unusual Pattern Detection, Outlier Detection

1. INTRODUCTION

Outlier detection, which aims at detecting abnormal observations, has became an interesting topic in data mining. Historically, outliers were considered noise that would adversely affect the output of the data analysis process and would need to be removed from the dataset. However, the deviation of outliers from other data may indicate interesting or fraudulent activities that require our attention instead of simply discarding them [6]. In practice, an outlier detection method can be used as an unsupervised technique for fraudulent activity detection, network instrusion detection and system monitoring. Outlier detection methods can be divided into two categories: statistical-based [6] and distance-based [4]. The statistical based approach can discover outliers by computing the probability of the observation from the underlying distributions of the dataset. However, those distributions are usually unknown in the data mining applications. In constrast, the distance based approach [4] can detect the outliers without knowing the underlying distributions. Recently, Breunig et al [2] has introduced the concept of a local outlier factor known as a density-based approach that could be used to detect outliers. However, current outlier detection approaches still have limitations. First, the data distribution in high dimensional space is sparse [1, 12] which makes most items appear as outliers. Second, the intersting observations may not be the top outliers. Those two problems lead to a very high false alarm rate in practice which makes the study of outliers non-trivial. The examples in the motivation section illustrate the scenario that some anomalous observations are undetectable by the distance-based or density-based approach.

In this paper, we introduce another type of anomalous observation and the support metric used to discover and rank the interesting anomalous observations. From our perspective, the anomalous observations can simply be noise due to data sparsity in a high dimensional space [1, 12]. In a large dataset, it is possible for some items to randomly deviate from the normal values making them outliers. For that noise, the possibility to form patterns is low due to the nature of randomness. Thus, the anomalous observations become interesting when they form patterns. The significance of those anomalous patterns is supported by the number of items in the pattern. One possible approach to discover the pattern of the outliers is to apply a clustering algorithm on the set of outliers. However, as shown in our experiments, when the outliers are removed from the dataset, the clustering on the outliers will not reveal the true patterns of the abnormal observations. Another alternative approach is to apply the density-based clustering algorithm on the entire dataset or the top outliers of the dataset. The small clusters are the anomalous patterns. This approach still has two limitations. First, the abnormal observations can be grouped into the bigger clusters. Second, the algorithms will return a high number of small clusters which makes the identification of the interesting patterns difficult.

The content of this paper is organized as follows. The mo-

tivation section presents the type of anomalous observations that can not be detected by LOF, a well-known local based outlier detection approach [2]. The problem is formally defined in the formal definition section. The framework for the algorithm, the running time complexity and the method for setting the parameters are described in the framework section. In the next section, the performance of the method and the comparisons with other possible alternatives are evaluated. The running time of the algorithm is evaluated in the performance section.

2. MOTIVATION

We consider an example in table 1a to illustrate another type of anomaly. The table consists of four items with three attributes A_1, A_2 and A_3. Table 3 shows the pair-wise distance between the items. In table 4, each column contains the statistics for the item in the column. The rows in the table show the k-distance [2] with k = 3, the average distance of all KNNs (3 nearest neighbors) and the LOF factor of the item. We assume that the set of four items is a subset of a larger dataset with the range of the attributes from 0 to 1000. Thus, normalization to compute LOF is unneccessary. According to table 4, X_1 and X_2 are highly ranked outliers due to their LOF score and X_4 is the lowest ranked outlier in the group. X_1 and X_2 are more unusual than X_4. A closer look at table 1a shows that this is not necessarily true. According to the table, the values for attributes A_2 and A_3 are actually almost similar for all the items. The values vary uniformly from 150 to 350. The items are similar in term of A_2 and A_3.

ID	A_1	A_2	A_3
X_1	8	250	300
X_2	9	250	250
X_3	10	350	150
X_4	50	300	200

(a) Example 1

ID	A_1	A_2	A_3
X_1	8	250	300
X_2	9	250	250
X_3	10	350	150
X_4	50	300	200
X_5	20	270	800
X_6	30	280	850
X_7	40	300	700
X_8	50	350	750
X_9	20	900	250
X_{10}	30	800	350
X_{11}	40	850	300
X_{12}	50	850	250

(b) Example 2

Table 1: Examples 1 and 2

For atttribute A_1, there are two distinct groups: the group of X_1, X_2 and X_3 with the mean of 9 and a group of X_4. The first attribute of X_4 deviates significantly from the other items in the table. It is five times greater than the average of group of X_1, X_2 and X_3. Despite this abnormality, X_4 is considered less of an outlier than X_1 and X_2 according to the definition of LOF. Even though the range of A_1 is smaller than those in A_2 and A_3, the deviation is significant. In this example, X_4 is unusual whereas X_1, X_2 and X_3 are normal.

We consider two possible alternative approaches to discover the unusual item X_4. The first approach is to compute the LOF on the random combinations of the attributes [9] with the expectation that the outliers that are overlooked

ID	A_1	A_2	A_3
X_1	8	250	300
X_2	9	250	250
X_3	10	350	150
X_4	50	300	200
X_{13}	40	250	350
X_{14}	30	250	350

(a) Example 3

ID	A_1	A_2	A_3
X_1	8	250	300
X_2	9	250	250
X_3	10	350	150
X_4	50	300	200
X_{13}	40	250	350
X_{14}	30	250	350
X_{15}	20	250	350
X_{16}	15	250	350

(b) Example 4

Table 2: Examples 3 and 4

Table 3: Distance Matrix

	X_1	X_2	X_3	X_4
X_1	0	50.01	180.29	119.43
X_2	50.01	0	141.42	81.74
X_3	180.29	141.42	0	81.24
X_4	119.43	81.74	81.24	0

in the entire feature space can be revealed in the subspace of the dataset. An example in table 1b shows that such a subspace may not exist. In this example, the dataset has 8 additional items. As we can see, the items form 3 distinct groups. X_1, X_2, X_3 and X_4 belong to one group. X_5, X_6, X_7 and X_8 belong to the same group for their high values in A_3. X_9, X_{10}, X_{11} and X_{12} are in another group for their high values in A_2. In the subspace of A_1 and A_2, X_4 is in the group of X_5, X_6, X_7 and X_8 and in the subspace of A_1 and A_3, X_4 is in the group of X_9, X_{10}, X_{11} and X_{12}. In those cases and other possible subspaces, X_4 is not unusual. *The example illustrates that such a subspace where the deviation of the anomalous observations is revealed may not exist.* Another approach is to compare the attributes for the items in the KNN list. In example 1, X_1, X_2 and X_3 are KNNs of X_4. Since the value on A_1 of X_4 is distinct from that of X_1, X_2 and X_3, it is tempting to conclude that X_4 is unusual. However, example 3 shows that it is not true. In this example, X_4 is normal with respect to X_{13} and X_{14} even though X_{13} and X_{14} are not in the KNN list of X_4. The conclusion based on the deviation of item from its KNNs on some attributes is incorrect.

The examples show the scenarios where anomalous records are undetectable by the typical distance-based outlier detection approach. In the next section, we formally define the problem and introduce a method to detect this type of anomalous observation.

3. FORMAL DEFINITIONS

We extend the definitions of the relational algebra to incorporate the concept of interval tuple and the operations

Table 4: Metrics

	X_1	X_2	X_3	X_4
k-distance	180.29	141.42	180.29	119.43
Mean	116.58	91.06	134.32	94.14
LOF Score	1.55	1.55	1.34	1.27

on the interval tuple. Recall that a relation R consists of n attributes $A_1, \ldots, A_n$, a tuple t of R is an ordered list of values corresponding to the attributes. The notation $t.A_i$ refers to attribute A_i of tuple t. Where applicable, we drop the attribute name and use the subscript, i.e. t_i, to refer to the attribute in order to simplify the definitions. In addition, we use the terms feature and attribute interchangably.

First, we introduce the concept of an interval-tuple and the interval based selection operation. An interval is a set of real numbers bounded by two end points, which can be represented by $[a, b]$. Definitions 1 and 2 provide a convenient way to query the relation R with respect to a given set of intervals. In example 1a, a relation R consists of X_1, X_2, X_3 and X_4. The operation $\sigma_I(R)$ and $\sigma_{I_1}(R)$ on $I \equiv \{[7, 10], [250, 250], [250, 350]\}$ return $\{X_1, X_2\}$ and $\{X_1, X_2, X_3\}$ respectively. $S \equiv \{X_1, X_2\}$ is covered by I.

DEFINITION 1. *An n-interval tuple I for a relation R is an ordered list of n intervals $I \equiv < I_1, \cdots, I_n >$, where each interval $I_i \subseteq dom(A_i)$. The i^{th} interval of I is refered as I_i. I_i is an interval of I on attribute A_i.*

DEFINITION 2. *An interval selection operation σ on an n-interval tuple I for a relation R, denoted by $\sigma_I(R)$, selects a subset S of the tuples from R such that each t in the subset satisfies the following condition: $t_i \in I_i$,$\forall I_i \neq NULL$. We say that S is covered by I, I is a cover interval-tuple of S and I_i is a cover interval of S. If I has only one interval, say I_i, $\sigma_{I_i}(R)$ can be used. In this case, we say $\sigma_{I_i}(R)$ is an interval selection operation on interval I_i for R.*

DEFINITION 3. *A cover interval-tuple I of a relation S is minimal if I_i is the smallest interval that covers S, $\forall I_i \neq \varnothing$.*

DEFINITION 4. *$\omega(R)$ is an interval operation on relation R that returns a minimal interval tuple I for R.*

The function ω in definition 4 is an inverse function of the function σ on the same relation. In the example above, $\omega(S)$ returns $J \equiv \{[8, 9], [250, 250], [250, 300]\}$.

DEFINITION 5. *α function between two interval-tuples on attribute i is defined by:*

$$\alpha(I_i, J_i) = \frac{\max(\inf I_i, \inf J_i) - \min(\sup I_i, \sup J_i)}{\min(\sup I_i, \sup J_i)}, \quad (1)$$

$$\text{if } I_i \cap J_i \equiv \varnothing$$

$$\text{and } \alpha(I_i, J_i) = 0 \text{ if } I_i \cap J_i \neq \varnothing \quad (2)$$

The function α measures the dissimarity between two interals on an attribute i based on the ratio of the difference between two intervals instead of the distance between two intervals. *When the ratio on all the attributes for two interval-tuple I and J is small, those two interval-tuples are considered close.* This concept is used to define the closeness between any two sets as below:

DEFINITION 6. *Two sets S and S' are close under α_c, denoted by $close(S, S'/\alpha_c) \equiv true$ if and only if $\alpha(\omega_i(S), \omega_i(S')) < \alpha_c, \forall i \in 1 \ldots n$.*

DEFINITION 7. *Given $P = \{S_i\}$, if $\forall S_i \in P$, there is at least one $S_j \in P$ such that S_i and S_j are close under α_c, we say $\{S_i\}$ forms a pattern P under α_c. We define $supp(P) = \sum |S_i|$ as the support of P.*

We consider example 4 in table 2b which has two sets $S_1 \equiv \{X_1, X_2, X_3\}$ and $S_2 \equiv \{X_4, X_{13}, X_{14}, X_{15}, X_{16}\}$. According to definition 7, S_1 and S_2 form a pattern. As we see, X_1 and X_4 are in the same pattern, even though X_1 and X_4 are dissimilar. Definition 7 implies the chain property that can result in very high support patterns. The property is used to define the normality and abnormality as follows:

DEFINITION 8. *We say a pattern P is normal under N_n if $supp(P) > N_u$*

DEFINITION 9. *We say a pattern P is anomalous under N_u if $supp(P) \leq N_u$*

When a set of observations can form a large pattern even though two items in the pattern are very different, those observations are normal. The set of observations that can not form a large pattern and are disconnected from other observations are anomalous. From those definitions, we have the following properties:

PROPERTY 1. *An observation t is normal if and only if t belongs to a pattern P such that $supp(P) > N_u$.*

PROPERTY 2. *An observation t is abnormal if and only if t does not belong to any pattern P such that $supp(P) > N_u$.*

PROPERTY 3. *The normal observations account for the majority of the dataset.*

With these properties, we can construct an unusual pattern detection framework based on a sampling method as follows:

- Given an item t, we create a sample of items that are likely to be similar to t by using some likelihood metric.
- Since most items are normal, it is likely that the items in the sample will form patterns.
- If t follows a pattern, there exists a pattern that contains t.
- If a pattern is highly supported, all the items in the pattern are disregarded as normal items.
- The low supported patterns are flagged as candidates for unusual pattern.
- We then try to match those unusual patterns with other normal patterns. If a match can be found for a candidate, it will be disregarded. Otherwise, the unmatched candidates are unusual patterns.

Property 1 allows the possibility of using a sampling technique to rule out a normal pattern without comparing an item with all other items in the dataset. In other words, it is possible to devise an algorithm to detect the unusual patterns in linear time. In this paper, we sample the dataset based on the attributes to discover the patterns according to the framework mentioned above.

We use an interval splitting function f^S to divide a set of tuples into smaller sets. First, the split function on a relation R divides each column into at least k intervals. Each interval is split into smaller intervals if the ratio of the difference between two consecutive ordered values from R on the same column in the interval is greater than α_s. The combinations of the intervals from different attributes constitute

the interval-tuples. We use $f^S(R/k, \alpha_s)$ to denote the set of interval-tuples returned from the split function on relation R.

The sampling is then performed as follows:

Given an interval I_i on attribute i, we select a sample R from dataset D such that $R \equiv \sigma_{I_i}(D)$. We then perform the split function f^S on R to obtain the set of interval-tuples $V \equiv f^S(R/k, \alpha_s)$. For each interval tuple $I \in V$, we create a set of tuples S from R where $S \equiv \sigma_I(R)$. If S does not satisfy the condition $|f_i^S(S/\alpha_s)| = 1, \forall i$, which means S can be divided into smaller sets, we split interval tuple I into smaller interval tuples by performing the split function $f^S(S/k = 1, \alpha_s)$ on S. The value of one for k means that we only split an interval when there is a change of the values in the interval. We say $\{S\}$ are the rules generated from sample R since all the items in the same rule are similar. The rules are used to create patterns. As we can see, an observation can be in different rules depending on the chosen sample. In general, the normal observations are in the normal patterns. In the algorithm, *we compare the unusual patterns against other normal patterns and try to merge the unusual patterns*. Therefore, the normal observations that are mistakenly flagged as anomalous will be grouped into the normal patterns.

In example 2, the split function on attributes A_1, A_2 and A_3 with $k = 2$ and $\alpha_s = 2$ returns the sets of intervals: $\{[8, 30], [30, 50]\}$, $\{[250, 300], [350, 350], [800, 900]\}$ and $\{[150, 300], [300, 350], [700, 850]\}$ respectively. The items are then grouped into 8 rules $\{X_1, X_2, X_3\}$, $\{X_4\}$, $\{X_5, X_6\}$, $\{X_7\}$, $\{X_8\}$, $\{X_9\}$, $\{X_{10}\}$, $\{X_{11}, X_{12}\}$. According to definition 7, we have the following patterns generated from the rules: $\{\{X_1, X_2, X_3\}\}$, $\{\{X_4\}\}$, $\{\{X_5, X_6\}, \{X_7\}, \{X_8\}\}$, $\{\{X_9\}, \{X_{10}\}, \{X_{11}, X_{12}\}\}$.

We introduce a score metric to determine the significance of the detected pattern. *The unusual patterns having the same support with higher score are more interesting*. Definition 10 defines the score between two interval tuples. The score of a pattern against another pattern is the smallest score between the two interval tuples of the patterns. As we see, the score of an unusual pattern is used to measure the deviation of an unusual observation from the normal observation. Thus, the score of an unusual pattern is the smallest score between it and the normal patterns.

DEFINITION 10. *o-score function between two interval tuples is defined by*

$$\textit{o-score(I,J)} \equiv \sqrt{\sum \alpha^2(I_i, J_i)} \tag{3}$$

DEFINITION 11. *The o-score between two patterns P and P' is defined by*

$$\textit{o-score(P,P')} = \min_{\forall S \in P \wedge S' \in P'} \textit{o-score}(\omega(S), \omega(S')) \tag{4}$$

DEFINITION 12. *Given a set of normal patterns $\{P_i\}$, the unusual score of a pattern P with respect to $\{P_i\}$ is defined by*

$$\textit{u-score(P)} = \min_{\forall P' \in \{P_i\}} \textit{o-score(P,P')} \tag{5}$$

4. FRAMEWORK

The outline for the algorithm is shown in algorithm 1. The algorithm consists of n rounds where n is the number of attributes. In the i^{th} round, the makeSamples function creates a set of samples from dataset D on attribute i. In this implementation, the samples are created as follows. The data set is sorted on attribute i and divided into chunks of the size N_c. Each chunk R is a sample from which to learn rules on attribute i.

The split function $f^S(R/k, \alpha_s)$ outputs a set V of interval-tuples. The set of rules $\{S\}$ are created for each interval-tuple $I \in V$. The makePatterns function merges all the rules and outputs a set NP of normal patterns a and set UP of unusual pattern candidates according to definition 7, 8 and 9. The unusual score for all unusual pattern candidates are also computed against set NP from line 11 to line 13.

The algorithm from line 3 to line 13 creates the normal patterns and unusual pattern candidates for each sample R. As discussed in the introduction, we need to rule out the normal patterns. Variable P at line 14 contains the set of unusual pattern candidates. For each candidate $p \in P$, p will be removed from P if there exists a normal pattern $p_n \in UP$ such that p and p_n are close under α_c (line 14). This is done since those candidates are shown to be normal in another pattern. For the first round, no action is performed on UP from line 15 to 16 and all the unusual pattern candidates are put into P. P contains the set of candidates for the first attribute. The items which do not belong to any pattern in this set of candiates are normal.

For the next rounds, the unusual pattern candidates will be removed from UP if they are not in P (line 15) because they were flagged as normal. The function at line 16 combines the unusual pattern candidates into larger patterns according to the close function under parameter α_c. The candidates which are normal after the combination are removed from P. The new unusual score of each new pattern is the lowest score from the candidates for which the new pattern is created. When the computation is finished for all rounds, all the unsual pattern candidates in P that do not match any normal pattern or do not have the support sufficient enough to be normal are unsual patterns.

The close and score functions are computed from the α function which requires the dividends to be non zero. For each round, we replace the zeros with the average of the next c items when the dataset is sorted on attribute i. The zeros can be replaced by 0.5 if the dataset contains only 0 and 1 for attribute i.

4.1 Parameter Setting

The algorithm consists of five parameters: α_c, α_s, k, N_c, N_u, which can be determined intuitively. N_u is a user-defined parameter indicating the size of a pattern to be considered unusual. N_c is the sample size. k is choosen based on the number of possible patterns in a sample. The number of patterns increases in the sample when k increases. However, if those patters are similar, they will be combined together eventually. The value of k does not impact the output significantly. Typically, k can be around 4 and N_c can be computed from $k \times N_u$. Besides k, α_s is used to split an interval if there is an abnormal change in the interval. The parameter α_c defines the cutoff point for the anomalous observations. Heuristically, we can choose α_c and α_s between 0.3 and 0.6.

4.2 Running-time Complexity

We denote $|NP|$, $|UP|$ and $|P|$ as the total number of items in NP, UP and P respectively. From line 5 to 9, it

Algorithm 1 Rule-based Pseudocode

```
1: procedure RuleBase(D)
2:     for all i ∈ n do
3:         U ← makeSamples(D, i)
4:         for all R ∈ U do
5:             V ← f^S(R/k, α_s)
6:             for all I ∈ L do
7:                 S ← makeRules(R, I)
8:                 put S into RP
9:             end for
10:            makePatterns(RP, NP, UP)
11:            for all p ∈ UP do
12:                compute o-score(p, NP)
13:            end for
14:            P ← matchNorm(NP, P)
15:            UP ← matchUnusual(UP, P)
16:            P ← combine(P, UP)
17:        end for
18:    end for
19: end procedure
```

takes $O(N_c)$ time to make the rules. The makePattern function needs to combine all the patterns which has a worst case of $O(N_c^2)$. The running time from line 11 to 13 is $O(|UP| \times |NP|)$. We can use the hashtables to store patterns for NP, UP and NP in order to execute the matching functions (line 14, 15) in linear time with respect to the number of items. The execution time for the two lines are $O(min(|NP|, |P|))$ and $O(min(|UP|, |P|))$ respectively. Since $|NP| + |UP| = N_c$, the execution time for them is less than $O(N_c)$. The running time for line 16 is $O(|UP| \times |P|)$. The total running time from line 3 to 16 is $O(N_c + N_c^2 + |UP| \times |NP| + N_c + |UP| \times |P|)$. Since N_c and $|UP| * |NP|$ are less than N_c^2, the formula can be reduced to $O(max(N_c^2, |UP| \times |P|))$. In general N_c^2 is small and does not grow with the dataset and $|UP| \times |P|$ is small on the average. The running time can be considered constant, $O(c)$.

There are n rounds. It takes $O(NlogN)$ time to sort the data according to the attribute of the current round. Each round has $\frac{N}{N_c}$ chunks and the execution time for each chunk is $O(c)$. Therefore, the total execution time is $O(c \times n \times \frac{N}{N_c} + n \times NlogN)$. When we replace $(\frac{c}{N_c} + logN)$ by ς , the formula can be written as $O(\varsigma \times n \times N)$. The formula shows that the algorithm can be executed in almost linear time.

5. EXPERIMENTS

We ran experiments using the KDD CUP 99 Network Connections Data Set from the UCI repository [11]. The data was compiled from a wide variety of intrusions simulated in a military network environment prepared by MIT Lincoln Labs. Each record has 42 features containing the network connection information. Among them, 34 features are continuous and 8 features are symbolic. The last feature labels the type of connection. About 80% of the data set are normal connections and 20% are attack connections. To make the experiment realistic, we create a new dataset with a very low number of attacks randomly drawn from the KDD Cup Dataset to test whether the small patterns would be discarded by the sampling method in a large dataset. The new dataset contains 95,174 normalized connections in total and the total number of attacks account for only 2.2% of the dataset. To make the experimental results less biased by the possible special characteristics of some types of attack, the data set contains 22 types of attack with the number of records for each attack type varying from 2 to 210. The attack with the largest size accounts for only 0.2% of the dataset. The details of the number of connections for each attack are shown in table 5.

Table 5: List of Attacks

Type	#attacks	Type	#attacks
pod	210	bufferoverflow	30
ipsweep	209	land	21
warezclient	203	warezmaster	20
portsweep	200	imap	12
nmap	200	rootkit	10
smurf	200	loadmodule	9
satan	197	ftp write	8
neptune	196	multihop	7
teardrop	179	phf	4
back	168	perl	3
guess passwd	53	spy	2

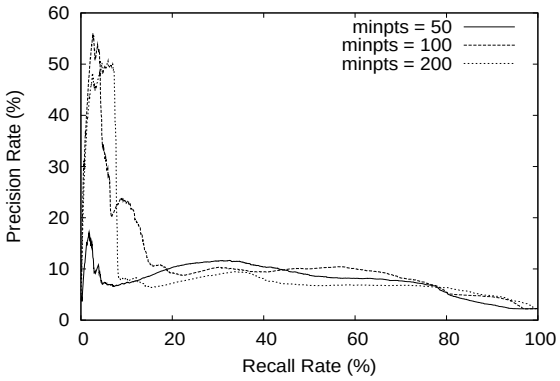

Figure 1: LOF Precision-Recall (PR) Curve.

5.1 Competing Methods

First, we run an oultier detection algorithm on the dataset to obtain the list of outliers. In this experiment, we use the LOF algorithm [2]. The records with high LOF score are considered as outliers. The score is computed based on the min_pts^{th} nearest neighbor [2]. The score is sensitive to the choice of min_pts. With small min_pts, the group of outliers may not be discovered. With large min_pts, the outliers in small clusters can be identified as normal because of the smoothing effect.

Figure 1 shows the LOF precision/recall curve for different values of min_pts. Generally, $min\_pts = 100$ performs better than the other two values of min_pts. As we see, LOF obtains the highest precision rate (56%) when the recall rate = 2.6%. When the recall rate reaches 20%, the precision rate drops dramastically to 10%.

Since the precision is low for high recall rate, we apply clustering algorithms on the outliers to group them into their corresponding attack types. First, we ran KMEAN on top outliers which account for 10% of the dataset. There are 40 clusters. We then filtered the output by removing clusters with the size greater than 250. There are 30 such clusters.

Table 6: Top outliers (10%) Clustered by KMEAN

ID	Rank	Size	Connections	Rate(%)
1	7	159	neptune	95.0
			portsweep	1.3
2	12	114	satan	0.9
			teardrop	89.5
			ipsweep	0.9
3	13	110	nmap	100.0
4	16	107	smurf	73.8
5	18	101	satan	1.0
			ipsweep	4.0
			portsweep	87.1

Table 7: Top outliers (10%) Clustered by SNN

ID	Rank	Size	Connections	Rate(%)
1	9	78	neptune	100.0
2	15	58	smurf	77.6
3	16	56	smurf	100.0

Ten of them have at least 50% of attack. Table 6 shows the corresponding rank of first five clusters that contain the attack connections. As we can see, 13 out of 18 top clusters do not contain any attack.

In the next experiment, we used a shared nearest neighbor (SNN) clustering algorithm to cluster the top outliers. SNN returns one cluster of size 3814 and 292 clusters of size less than 143. There are 19 clusters with the size from 50 to 143. Table 7 shows top three clusters that contains the attack connections. The first attack cluster ranks 9^{th} and its size is 78. In those two experiments, the attack connections are not shown as strong unusual patterns.

Since the clustering on the top outliers could not detect all the top ten groups of attack. We want to cluster the entire dataset with the hope that those groups can be discovered. It is difficult to set the parameter k for KMEAN in this large dataset. With large k, KMEAN will return many clusters whereas small values of k may group the normal connections with attack connections. Therefore, we used SNN for this experiment since it can discover small clusters with different densities in large datasets [5] without requiring prior number of clusters as the input. The algorithm returns 131 clusters with the size from 50 to 250. Among them, there are six clusters that contain attack connections (see table 8). The first cluster that contains the attack connections is ranked 35^{th}. As we can see, the attack and normal connections are divided into smaller clusters. The attack patterns are not shown clearly in this experiment. The result shows that even though SNN can discover small clusters, they will return many of them.

Table 8: SNN on the entire dataset

ID	Rank	Size	Connections	Rate(%)
1	35	112	nmap	100.0
2	57	89	warezclient	73.0
			rootkit	1.1
3	74	79	neptune	100.0
4	90	65	ipsweep	100.0
5	112	59	smurf	78.0
6	118	58	smurf	100.0

5.2 Our Method

In the next experiment, we ran our method on the dataset to discover the unusual patterns. We set the parameters according to the parameter setting from the algorithm section (see table 9).

Table 9: Parameter Setting

N_u	250	α_c	0.5
k	4	α_s	0.5
N_c	1200		

Table 10: Top 10 Unusual Patterns Ordered by Size

Rank	Size	Score	Connections	Rate(%)
1	243	24.1	smurf	79.4
			normal	20.6
2	192	43.6	nmap	100
3	170	12.9	normal	100
4	169	203.5	satan	100
5	150	104.1	neptune	100
6	129	26.1	ipsweep	100
7	114	14.2	back	100
8	84	6.5	warezclient	100
9	72	15.6	normal	100
10	67	107.7	portsweep	100

The algorithm returns 20 unsual patterns with the size of at least 50. Table 10 shows top 10 unusual patterns by size. The size of those patterns varies from 67 to 243. The first two patterns are of attack type. Seventy percent of those patterns contain 100% of attacks. The other 9 smaller patterns containing attacks are shown in table 11. According to table 10, we see that satan, neptune and portsweep follow strong patterns. In the table, we see that warezclient attacks also follow a pattern but its score is low (6.5) in relative to those attack types, which means that its pattern is slightly different from normal patterns.

As mentioned above, the data set contains 10 attack patterns with the size of at least 100. In our method, eight of ten are identified in the top unusual patterns (by size).

Table 13 shows the recall rate for each attack type found in table 10. With the low false alarm rate, we still get a high recall rate.

We then take all the unusual patterns with the size of at least 50 and order them by score. Table 12 shows the ranking, score and attack type of the patterns. According to

Table 11: Other Unusual Patterns Contain the Attacks

Rank	Size	Score	Connections	Rate(%)
13	66	11.8	teardrop	100
14	64	54.	pod	100
20	50	48.8	pod	100
22	48	83.1	guesspwd	100
23	44	156.6	neptune	100
33	31	20.0	teardrop	100
35	29	30.0	back	100

Table 12: Unusual Patterns Ordered by Score

Rank	Type	Score	Size
1	satan	203.5	169
2	portsweep	107.7	67
3	neptune	104.1	150
4	pod	54.6	64
5	pod	48.8	50
6	nmap	43.6	192
7	normal	29.6	54
8	normal	26.2	63
9	ipsweep	26.1	127
10	normal	25.9	59
11	smurf	24.1	243
12	normal	15.6	72
13	normal	14.9	64
14	back	14.2	114
15	normal	12.87	170
16	teardrop	11.8	66
17	normal	10.5	67
18	normal	9.7	56
19	warezclient	6.6	84
20	normal	4.2	67

Table 13: The Recall Rate of the Attack Types in Top 10 Patterns

Type	Rate	Type	Rate
smurf	96.5	back	67.9
nmap	96	ipsweep	61.2
satan	85.8	warezclient	41.4
neptune	76.5	portsweep	33.5

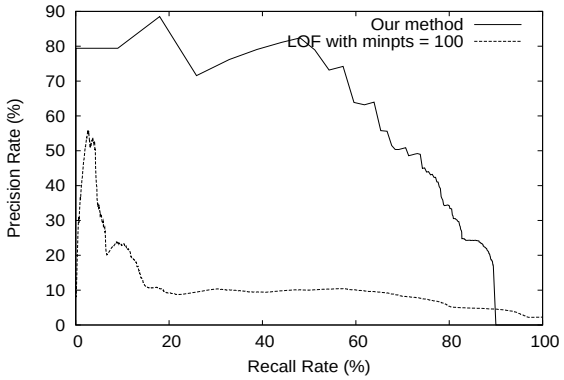

Figure 2: Precision-Recall (PR) Curve.

the table, our method correctly identifies some of the attacks in the first six unusual patterns. All of these patterns have the detection rate of 100%. Among them, the attack type of Satan has the highest score which is 203.5. The score of the first normal connection pattern in the table is only 29.6 and its size is only 54.

In ranking either by size or by score (after the patterns with very low size are pruned), we can see that the attack types of satan, portsweep, neptune, and nmap can be discovered well by our approach. The results imply that these types of attack follows some patterns which are strongly different from normal connections.

Figure 2 shows the PR curve of our method versus LOF. The figure shows that our method yields a precision rate of 80% at very low recall rates. The method outperforms the LOF approach. Since we quickly prune the patterns with the unusual score below the cutoff threshold, the attacks with very low unsual score were removed from the output. That is why the figure only shows the recall rate up to 90%. However, this does not affect the result much since the precision rate is usually low at this recall rate.

According to the experiments, the following interesting observations are found. *Most unsual patterns with normal connections do not form highly supported patterns.* Even though they may have high scores, they appeared as very low support patterns, whereas the attack connections form patterns with high support. *The normal connections that can form patterns with high support tend to have a very low score.* Few discovered unusual patterns have mixed results. Another important observation is that the number of unusual patterns with high support is very small.

In conclusion, we have performed a variety of experiments with different combinations of outlier detection and density-based clustering algorithms. The precision is low when the recall rate increases to 20%. For the clustering algorithms, the normal connections were also grouped into small clusters. However, our algorithm grouped the attack and normal connections almost correctly according to their connection type even though there are 22 types of attacks and their size is small. They are shown clearly as unusual patterns in terms of size and score.

6. PERFORMANCE

We implement a memory-based version of the algorithm in Java. For each round, the dataset can be sorted in $O(NlogN)$ time. We use a hashtable data structure to store the list of unusual candidate items so that the matching functions, matchNorm and matchUnusual, can be performed in constant time. At first, we ran the program with different initial attributes. The number of unusual candidates vary from 30K to 60K. During the first round, we don't combine the unusual patterns, therefore, the performance is not affected. In the next few rounds, the matching functions reduce the number of unusual items dramatically before the combining step.

Figure 3 shows the performance of the algorithm with different random orders of the attributes. According to the figure, we see that the order of the attributes does not affect

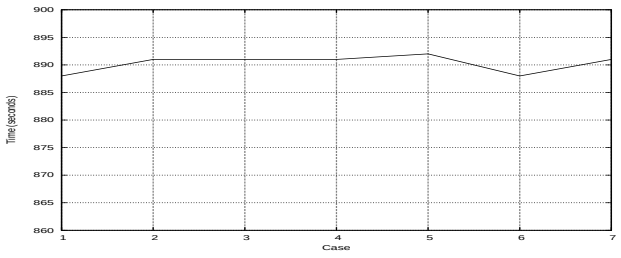

Figure 3: Running time for the different orders of the attributes.

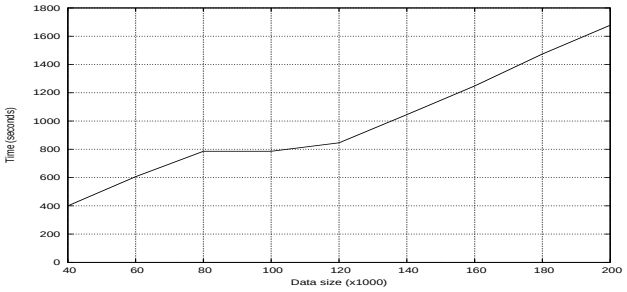

Figure 4: Running time for the algorithm.

the running time significantly. Also, the results are almost the same for different orders of the attributes. The attack connections are consistently in the top unusual patterns.

In the next experiment, we ran the program on the KDD dataset with the size varying from 40K to 200K. Figure 4 shows that the execution time of the program is linear with the growth of the data size. By replacing the memory-based hashtable data structure and merge sort algorithm with the disk-based versions, the algorithm can be used for any large dataset.

7. RELATED WORK

Outlier detection has been extensively studied in the field of statistics [6]. The method relies on using the underlying distribution of the dataset to detect the outliers. The limitations of the statistical-based approach is that the underlying distribution is usually unknown and that approach does not perform well in high dimensions. The distance-based approach can discover outliers without knowing the underlying distribution [4, 2]. However, we have shown that there are cases where the anomalous observations can not be detected by the distance-based approach.

Recently, a method [3] was introduced to discover anamolous records in categorical datasets by performing the conditional probability tests on combinations of the attribute values. The test requires that the values must be discretized into the set of values. Hence, it is more suitable for the categorical dataset.

Density-based clustering methods cluster the dataset based on the local density of the nearby items. The nearby items with the same density are clustered together [5, 7]. Density-based clustering can discover the clusters with different sizes and shapes [5]. The main difference between this approach and our unusual pattern detection is that this method focuses on clustering the dataset, whereas our approach focuses on discovering the unusual pattern. The items that are normal are quickly removed from the learning process. As a result, our algorithm can run faster without dividing the data set into small clusters.

8. CONCLUSION

We have introduced a fast detection algorithm that can discover anomalous observations. From our perspective, the unusual items become interesting when they form small patterns. By using the properties of normal and unusual observations, we have developed a sampling method to rule out the normal observations in order to detect unusual patterns. The algorithm generates the rules from the samples. We then combine the rules into patterns. When a pattern gains high support, the pattern is normal and all the items in the pattern are discarded. The unusual patterns are ranked by their support. In the experiments, we have compared our algorithm with other possible alternatives, namely outlier detection and clustering, to discover unusual patterns. According to the results, our approach yields the highest detection rate with the most unusual items grouped into the correct corresponding patterns. We have also introduced the score function to measure the degree of deviation of the unusual pattern from the normal patterns. The running time complexity of the algorithm is $O(\zeta \times d \times N)$ where ζ is a constant. The experiments confirm that our algorithm can run fast in near linear time.

9. REFERENCES

[1] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *ICDT '99: Proceeding of the 7th International Conference on Database Theory*, pages 217–235, London, UK, 1999. Springer-Verlag.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, 2000.

[3] K. Das and J. Schneider. Detecting anomalous records in categorical datasets. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 220–229, New York, NY, USA, 2007. ACM.

[4] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 392–403, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[5] L. Ertöz, M. Steinbach, and V. Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the third SIAM international conference on data mining*, pages 47–58. Society for Industrial and Applied, 2003.

[6] D. Hawkins. *Identification of outliers*. Chapman and Hall, London, 1980.

[7] R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, C-22(11):1025–1034, 1973.

[8] F. Korn, B.-U. Pagel, and C. Faloutsos. On the 'dimensionality curse' and the 'self-similarity blessing'. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):96–111, 2001.

[9] A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166, New York, NY, USA, 2005. ACM.

[10] J.-C. C. E. E. N. M. Marco-Antonio Balderas, Fernando Berzal. Discovering hidden association rules. In *KDD '05: Internal workshop on data mining methods for anomaly detection*, pages 357–361. ACM, 2005.

[11] C. B. D. Newman and C. Merz. UCI repository of machine learning databases, 1998.

[12] U. Shaft and R. Ramakrishnan. Theory of nearest neighbors indexability. *ACM Trans. Database Syst.*, 31(3):814–838, 2006.